Monadic Second-Order Logic and Pomset Languages

Tobias Kappé

Cornell University

CPP 2021 — Lightning Talks

Verification

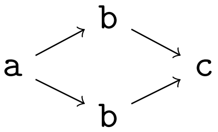
$$P \models \phi$$

$$[\![P]\!] = \left\{ \begin{array}{c} \mathtt{a} \to \mathtt{b},\ \mathtt{a} \to \mathtt{b},\ \mathtt{b} \to \mathtt{c},\ \mathtt{b} \to \mathtt{c} \end{array}, \ldots \right\}$$

$$\forall x.\ \lambda(x) = \mathtt{b} \implies \exists y.\ x \leq y \wedge \lambda(y) = \mathtt{c}$$

Verification

$$P \models \phi$$

$$[\![P]\!] = \Big\{$$ 

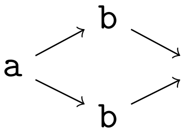
$$, \ldots \Big\}$$

$$\forall x.\ \lambda(x) = \mathtt{b} \implies \exists y.\ x \leq y \wedge \lambda(y) = \mathtt{c}$$

Verification

$$P \models \phi$$

$[\![P]\!] = \Big\{$



$, \ldots \Big\}$

$$\forall x.\ \lambda(x) = \mathtt{b} \implies \exists y.\ x \leq y \land \lambda(y) = \mathtt{c}$$

Monadic Second-Order Logic

Theorem (Büchi, Elgot, Trakhtenbrot)

Let $\mathcal{L}$ be an MSO-definable language of words.
We can construct a finite automaton for $\mathcal{L}$.

Theorem (Kuske)

Let $\mathcal{L}$ be an MSO-definable language of pomsets.
There exists a finite bimonoid that recognises $\mathcal{L}$.*

Monadic Second-Order Logic

Theorem (Büchi, Elgot, Trakhtenbrot)

Let $\mathcal{L}$ be an MSO-definable language of words.
We can construct a finite automaton for $\mathcal{L}$.

Theorem (Kuske)

Let $\mathcal{L}$ be an MSO-definable language of pomsets.
There exists a finite bimonoid that recognises $\mathcal{L}$.*

Objectives

- Constructive translation of formulas to bimonoids.
- Proof that computed bimonoid accepts the same pomsets.
- Extraction of bimonoid code for use in model checking.

Objectives

- Constructive translation of formulas to bimonoids.
- Proof that computed bimonoid accepts the same pomsets.
- Extraction of bimonoid code for use in model checking.

Objectives

- Constructive translation of formulas to bimonoids.
- Proof that computed bimonoid accepts the same pomsets.
- Extraction of bimonoid code for use in model checking.

Encoding in Coq

Pomsets

```
Record pomset (A: Type) := MkPomset {
    pomset_carrier: Type;
    pomset_order: pomset_carrier -> pomset_carrier -> Prop;
    pomset_labeling: pomset_carrier -> A;

    (* + pomset laws *)
}.
```

Encoding in Coq

Formulas

```
Inductive formula (A TP TS: Type) :=
| Before (x y: TP)                              (*  x ≤ y  *)
| Member (x: TP) (X: TS)                        (*  x ∈ X  *)
| Label (x: TP) (a: A)                          (*  λ(x) = a  *)
| Disjunction (l r: formula)                    (*  l ∨ r  *)
| Negation (inner: formula)                     (*  ¬inner  *)
| ExPos (inner: formula A (TP + unit) TS)       (*  ∃x. inner  *)
| ExSet (inner: formula A TP (TS + unit))       (*  ∃X. inner  *)
.
```

Encoding in Coq

Satisfaction

```
Fixpoint satisfies
    {A TP TS: Type}
    (f: formula A TP TS)
    (u: pomset A)
    : Prop
:=
    (* snip *)
.
```

```
Fixpoint implementation
    {A TP TS: Type}
    (f: formula A TP TS)
    : bimonoid
:=
    (* work in progress *)
.
```

Encoding in Coq

Correctness

```
Lemma correctness
    {A: Type}
    (f: formula A Empty_set Empty_set)
:=
    forall u: pomset A,
        bimonoid_eval (implementation f) u = true
        <-> satisfies f u
.
Proof.
    (* work in progress *)
Admitted.
```

Thoughts

- Constructive nature is nice for this proof.
- Still learning; currently grokking finite sets...
- Let's talk! `tobias@kap.pe`

Thoughts

- Constructive nature is nice for this proof.
- Still learning; currently grokking finite sets...
- Let's talk! `tobias@kap.pe`

Thoughts

- Constructive nature is nice for this proof.
- Still learning; currently grokking finite sets. . .
- Let's talk! `tobias@kap.pe`